优秀手游成长之道之《暗黑破坏神：不朽》程序篇

Adele(阿迪拉·阿力木江) 等 2022.07.28 10:21 3428 58 12个资源

暴雪×网易联合出品的《暗黑破坏神®:不朽™》作为暗黑系列的全新作品,不仅传承了经典暗黑画风和恢弘世界观,还原了畅爽战斗体验和沉浸的探索乐趣。

目录

精华分享

《暗黑破坏神：不朽》程序成长之道 序言

g67《暗黑破坏神：不朽》 messiah server集成msgpack（纯C+

自定义监控服务器指标——G67《暗黑破坏神：不朽》是怎么做

又想网络质量好，又想省流量——G67《暗黑破坏神：不朽》

G67《暗黑破坏神：不朽》 Android so加固方案

G67《暗黑破坏神：不朽》服务器增量（局部）热更新（reloa

Bent Normal是否适合G67

记录G67的安卓Vulkan适配

AMD FSR in G67

高通VRS在G67的应用

一个神秘的G67兼容性问题

记录G67的安卓Vulkan适配

梁骏图 2022.05.21 22:57 423 10 0 查看原文

本文仅面向以下用户开放，请注意内容保密范围

查看权限：互娱正式-公开

> 本文记录了开启安卓的Vulkan管线，通过MultiPass降低Arm芯片的机器的GPU带宽，并且通过减少Subpass数量的优化减
> 下5%的GPU消耗。同时还在GLES3和Vulkan都接入了ARM的ASTC RGBA8 Decode拓展，优化对ASTC贴图的采样，获得
> 消耗的提升。

记录G67的安卓Vulkan适配

1 概述

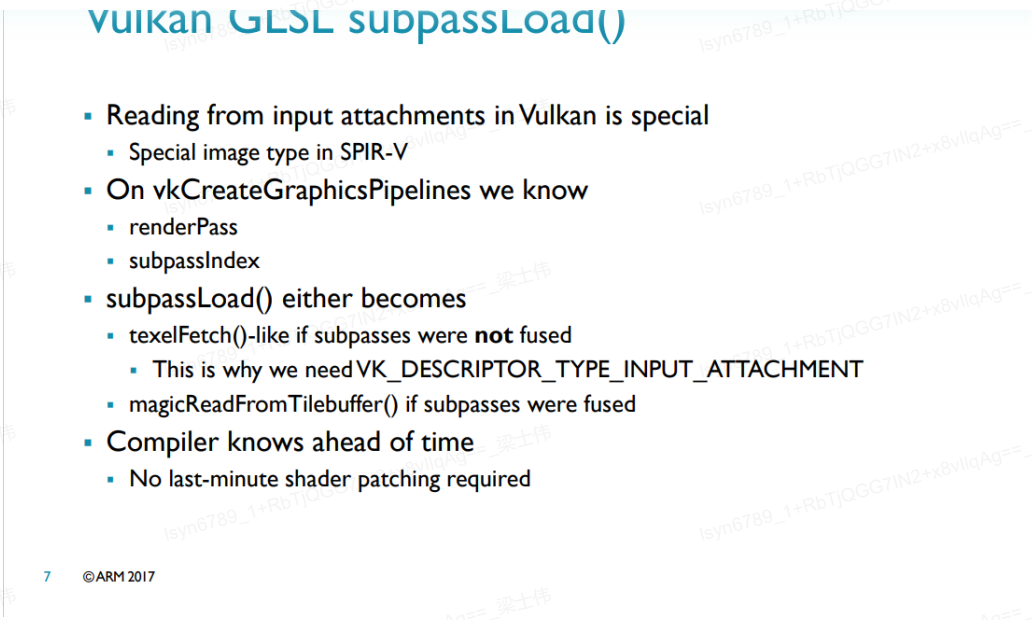
本文记录了开启安卓的Vulkan管线，通过MultiPass降低Arm芯片的机器的GPU带宽，并且通过减少Subpass数量的优化减少了Vulkan
耗。同时还在GLES3和Vulkan都接入了ARM的ASTC RGBA8 Decode拓展，优化对ASTC贴图的采样，获得10%GPU消耗的提升。

2 问题背景

移动端GPU都采用Tile Based架构的一个重要原因是为了降低带宽，因为它非常大的影响了GPU的发热和功耗。
而我们项目只有Deferred管线了，不停的采样GBuffer对带宽影响非常的大。在苹果的Metal和高通GPU的ES3上都支持FrameBufferFe
分的Pass跑在OnePassDeferred上，极大的减少了带宽。但是在Arm的GPU上则没那么幸运，只能老老实实的去采样GBuffer。而在与
流中得知，也没有要支持FrameBufferFetch的打算，但是他们也给我们指了一条路，那就是试试Vulkan。

3 什么是Vulkan MultiPass?

尝试Vulkan主要是为了应用MultiPass，Arm在GDC 的分享Vulkan Multipass at GDC 2017对它的原理已经解析的很清楚了，可以认为
构上跟ES3的FrameBufferFetch有着一样的底层实现。



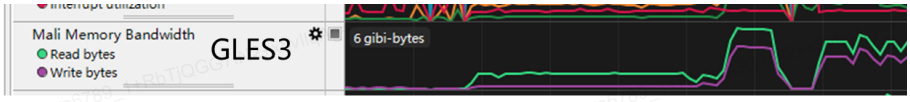
4 开启Vulkan

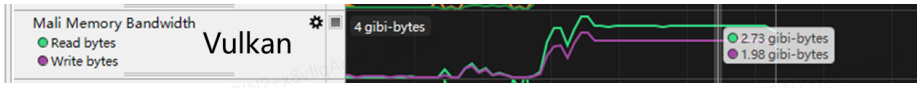
带宽降了多少?

在Messiah2020.2的Vulkan中已经支持了Subpass，但是我们项目的引擎在各种魔改后，已经开不起来。花了不少时间把各种Crash都
试一下看下效果。
这里用的是Arm Stream Line来在线获取带宽数据，带宽换算回一帧的大小：

- 华为p40 pro/麒麟990/60帧
  - GLES3一帧要167M
  - Vulkan一帧只要88M

- 华为Mate40 pro+/麒麟9000/60帧
  - GLES3一帧要127M
  - Vulkan一帧只要78M





可见带宽数据得到明显的优化，下降了40%到50%。可以进行下一步，让QA测试了一下性能。

帧率怎么样

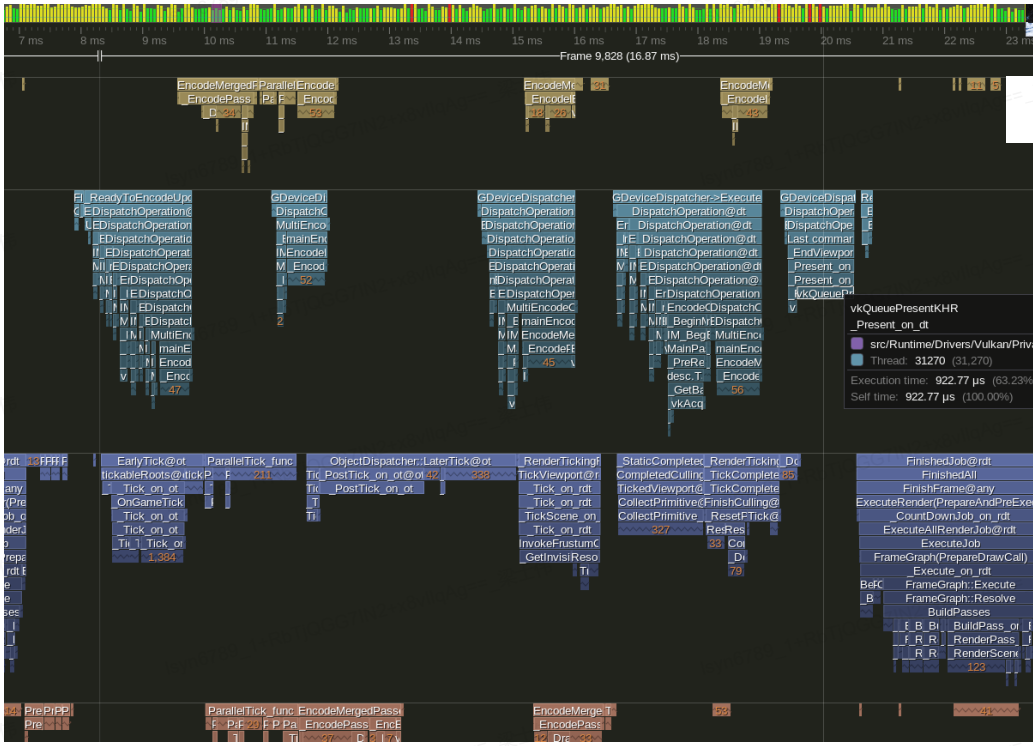
在两个MMO场景分别测试了30帧和60帧的性能：

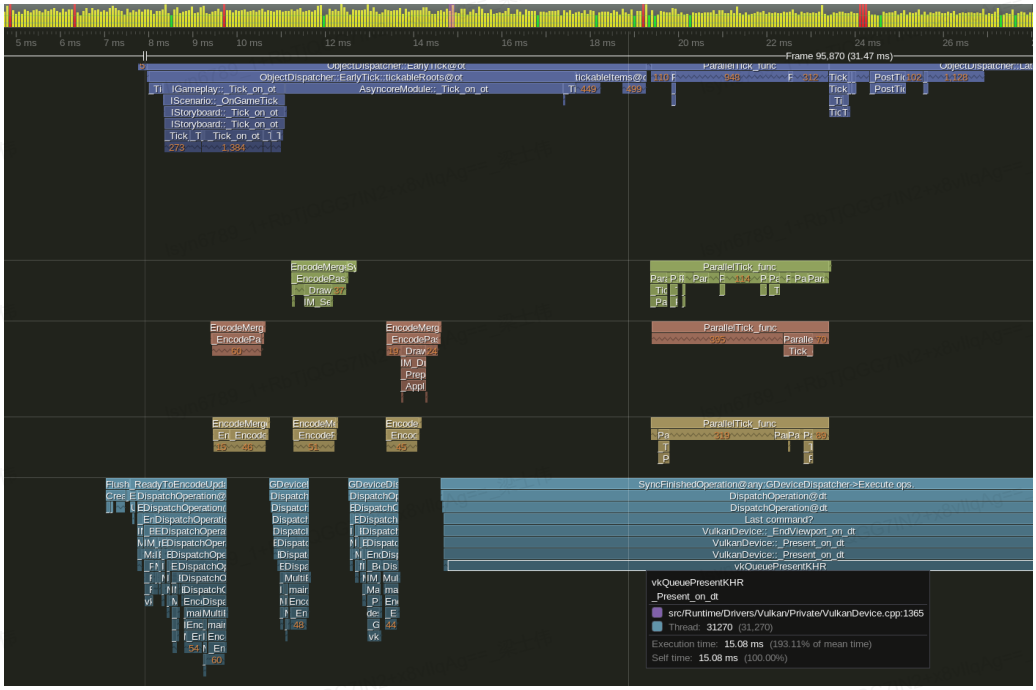
	帧率	扎瓦因山地	墓园
天玑1000	30	41	29.4
海思 Kirin 990	60	42.5	43.1
海思 Kirin 9000	60	48.8	
MTK 天玑1200	60	49.24	

还没达到发热测试60帧的要求

可见大概800秒后帧率开始掉，应该是发热降频了。在对比一下Tracy数据：

发热前



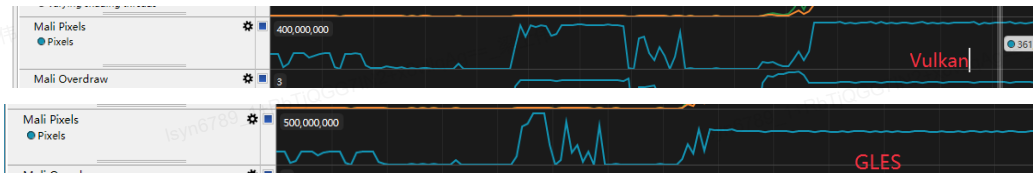


可见发热Present的时间变长了不少，验证了前面的发热降频的想法：虽然Command已经提交了，但是发热降频之后GPU渲染不过来。

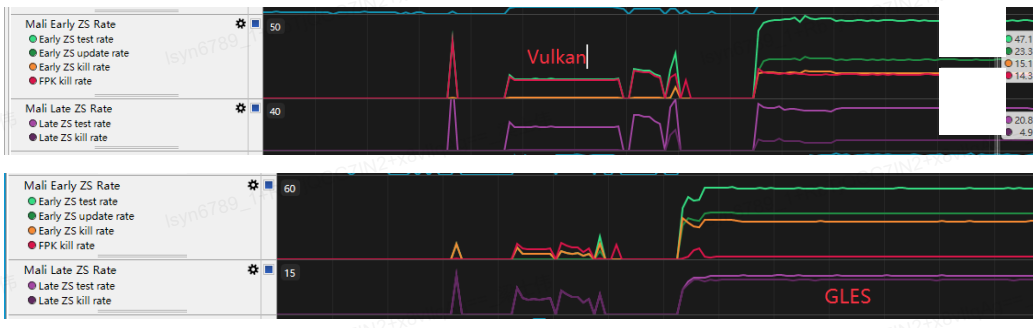
另外通过Tracy还发现了一个小问题。当前是等待超过128个DrawCall才会vkQueueSubmit，但这样对一下小的Pass不太友好，就是小之后不会提交渲染，在等待的时候GPU就会空闲着。而我们每一帧开头的小Pass还不少，有诸如体积雾，风力场，CSMScroll这些，所以待去掉好些。

再从硬件的角度看下Vulkan发热的问题，通过Arm StreamLine获取天机1200的GPU数据，并且发现Vulkan上的一个奇怪的问题。
对比Vulkan和GLES3在渲染相同的内容(墓园出生点)时的硬件指标，发现Vulkan的GPU Cycles数要比GLES3多不少，代表GPU压力更大

Vulkan的Pixels统计会比GLES的高出不少，多了19%



进一步看Early ZS Test和Late ZS Test也有区别：
Vulkan有更少的quads进入Early ZS Test，并且被Kill掉的也更少；另外Vulkan进入Late ZS Test的quads也更多



带宽指标Vulkan是好非常多的，少了33%的读写带宽，少了52%的Tile Buffer Write。
但是在两者 visible primitives 接近 (330k)的情况下， Vulkan的rasterized quads 却增加 63%
最终导致Vulkan的GPU Active Cycles从6,994,335涨到了10,678,109，增加了53%。

MTK的工程师初步排查，发现Vulkan和GLES两边的RT数与分辨率不太一样。RenderDoc截帧一下，确实是这样，原来是在不支持fram
时，为了优化带宽，渲染管线的顺序会有些不太一样。但是当渲染管线调整到一模一样后再测试，Vulkan的GPU Active Cycles仍然比G
以这应该不是主因。

于此同时也与MTK合作，测试了新芯片上的性能。

MTK的测试

测试结果如下：

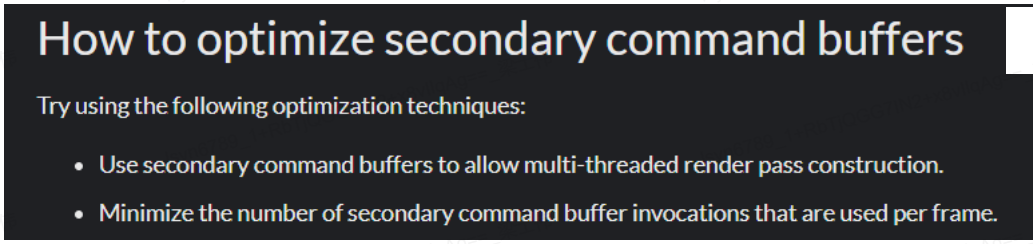
	A	B	C	D	E
1				帧率	功耗mA
2	天机9000	960P	Vulkan	59.2	1507
3			GLES3	55.3	1647
4	天机9000	720P	Vulkan	59.1	1209
5			GLES3	59.3	1400
6	天机8100	720P	Vulkan	58.8	971
7			GLES3	59.1	1064
8	天机1200	720P	Vulkan	56.3	1250
9			GLES3	52.8	1292
10	新平台	720P	Vulkan	58.7	1104
11				57	1160
12					

以上数据基于泰摩的标准60帧测试。

可见整体而言，各芯片在Vulkan的表现优于GLES3。特别是功耗，在所有测试中Vulkan都更少，功耗最多减少了14%(在天机9000的72
功于带宽的减少。而帧率上，720P两者表现差不多，但是压力增大到960P时，Vulkan的表现更优。

此外这次测试天机1200的数据确实也没有预想的好，不过也发现了一些问题。

首先是CommandBuffer的问题。由于Vulkan是多线程Encode的，而我们安卓引擎中Encode的线程数等于CPU大核的数量，但是在4个
现了5个CommandBuffer。通过排查发现是在把Game Thread和Device Thread合并的时候出来的一个Bug，遂改正。Arm Mali GPU
Developer Guide Version 2.2上也指出过当前Mali的GPU不能直接从secondary command buffer执行指令，所以是有额外消耗的，
宜过多。所以既要用它来做多线程Encode，但是又要注意控制它的数量：



而MTK的工程师给的建议是CommandBuffer的总数不要多余4个。

其次是多线程Encode的Shared线程在某些机器跑在了小核心上，导致Encode的时间变长。并且小核的频率也高起来，这也是天机1200
的功耗没有比GLES3的少多少的原因之一。但是我们在线程创建的时候是有UseBigCore，然而系统仍然调度到小核上，对于这个问题，
没有什么好的建议，各个手机厂商会自己修改调度算法。对于这种机器，我们尝试了把多线程Encode关闭掉，不见得完全是正优化。

华为mate40 pro+	海思 Kirin 9000	60	Vulkan	48.8
			Vulkan(无多线程)	52.1
oppo Reno6 pro	MTK 天玑1200	60	Vulkan	49.24
			Vulkan(无多线程)	48.8

后来再看这个问题时，感觉还可以优化一下。当前是把一个Pass中的所有DrawCall平均分到不同的线程，但是这里没有考虑到不同线程
以改造成生产者消费者的方式，让处理能力更高的线程处理更多的Drawcall。不过由于这个改动还是比较大，赶不上引擎封版，所以还

选择了优先排查前面Vulkan的GPU Active Cycles更多的问题，再这期间对渲染管线做了许多的简化和对比，最终把问题定位到Subpas

罪魁祸首是Subpass

具体表现是在OnePassDeferred中，只要通过vkCmdNextSubpass()切换到下一个Subpass，GPU Active Cycles就会爆涨
掉，Vulkan在GBuffer后每加入一个pass的GPU Active Cycles的涨幅与GLES3是非常接近的。但是这只能是个测试，因为
画面都不对了，这是由于GPU需要它在并行的Pass之间做同步。
那么是不是我们的Subpass写的有问题?
Vulkan需要在创建RenderPass的时候就指定所有Subpass的信息：
通过在VkRenderPassCreateInfo中指定VkSubpassDescription和VkSubpassDependency

```
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags       flags;
    VkPipelineBindPoint             pipelineBindPoint;
    uint32_t                        inputAttachmentCount;
    const VkAttachmentReference*    pInputAttachments;
    uint32_t                        colorAttachmentCount;
    const VkAttachmentReference*    pColorAttachments;
    const VkAttachmentReference*    pResolveAttachments;
    const VkAttachmentReference*    pDepthStencilAttachment;
    uint32_t                        preserveAttachmentCount;
    const uint32_t*                 pPreserveAttachments;
} VkSubpassDescription;
```

```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDependency {
    uint32_t                srcSubpass;
    uint32_t                dstSubpass;
    VkPipelineStageFlags    srcStageMask;
    VkPipelineStageFlags    dstStageMask;
    VkAccessFlags           srcAccessMask;
    VkAccessFlags           dstAccessMask;
    VkDependencyFlags       dependencyFlags;
} VkSubpassDependency;
```

其中VkSubpassDescription指定个各个Subpass输入和输出，以及它们的VkImageLayout。
不同的VkImageLayout会影响Image的访问效率，因为它指定的是Image的数据在内存中的布局。假如读取是一行一行的读，但是存储的，这会降低Cache的命中率。另外一个Image可能在一个渲染管线中充当不同的角色，例如在前面的Pass作为ColorAttachment，而被当做贴图由Shader来采样，所以我们需要在渲染管线的合适位置插入PipelineBarrier来做ImageLayout的转换。

vkCreateRenderPass	vkCreateRenderPass([VkAttachmentDescription[5], [[[0, 1, 2, 3], 4], [[UINT32_MAX, 1, UINT32_MAX,
device	**Device 10**
CreateInfo	VkRenderPassCreateInfo()
sType	VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO
pNext	NULL
flags	VkRenderPassCreateFlagBits(0)
attachmentCount	5
pAttachments	VkAttachmentDescription[5]
subpassCount	4
pSubpasses	VkSubpassDescription[4]
[0]	VkSubpassDescription()
[1]	VkSubpassDescription()
[2]	VkSubpassDescription()
flags	VkSubpassDescriptionFlagBits(0)
pipelineBindPoint	VK_PIPELINE_BIND_POINT_GRAPHICS
inputAttachmentCount	4
pInputAttachments	VkAttachmentReference[4]
colorAttachmentCount	4
pColorAttachments	VkAttachmentReference[4]
pResolveAttachments	VkAttachmentReference[0]
pDepthStencilAttachment	VkAttachmentReference()
attachment	4
layout	VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL
preserveAttachmentCount	0
pPreserveAttachments	uint32_t[0]
[3]	VkSubpassDescription()
dependencyCount	4
pDependencies	VkSubpassDependency[4]
pAllocator	NULL
RenderPass	**Render Pass 1363**

有一个问题是DepthStencil的ImageLayout一直是VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL，而Arm Mali G Practices中指出在GBufferPass之后，如果不再修改DepthStencil，设置ImageLayout为ReadOnly可以提升性能。

How to optimize the use of multipass rendering with Vulkan

Try using the following optimization techniques:

- Use multipass.
- Use a 128-bit G-buffer budget for color.
- Use by-region dependencies between subpasses.
- Use DEPTH_STENCIL_READ_ONLY image layout for depth after the G-buffer pass is done.
- Use LAZILY_ALLOCATED memory to back images for every attachment except for the light buffer, which is the only texture that is written out to m
- Follow the basic render pass best practices, with LOAD_OP_CLEAR or LOAD_OP_DONT_CARE for attachment loads and STORE_OP_DONT_CAR

而我们在GBufferPass之后Depth是不变了，但是Stencil会修改，所以可以设置为VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL。
说不定就是它导致前面的发现：Vulkan管线的Early ZS Test和Late ZS Test都比GLES3的要差。
然而改正过来后，并未从Arm StreamLine看到有变化，所以应该不是它。

再看另一个VkSubpassDependency，它指明了各Subpass之间执行的先后和读写访问的依赖。
在我们的管线中并没有分析各Pass之间的真实依赖，只是简单的下一个Subpass依赖上一个Subpass。

Resource Initialisation Parameters

Parameter	Value
pSubpasses	VkSubpassDescription[4]
dependencyCount	4
pDependencies	VkSubpassDependency[4]
[0]	VkSubpassDependency()
[1]	VkSubpassDependency()
srcSubpass	0
dstSubpass	1
srcStageMask	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
dstStageMask	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT | VK_PIPELINE
srcAccessMask	VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_ACCESS_DEPTH_S
dstAccessMask	VK_ACCESS_INPUT_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_A
dependencyFlags	VK_DEPENDENCY_BY_REGION_BIT
[2]	VkSubpassDependency()
srcSubpass	1
dstSubpass	2
srcStageMask	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
dstStageMask	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT | VK_PIPELINE
srcAccessMask	VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_ACCESS_DEPTH_S
dstAccessMask	VK_ACCESS_INPUT_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCES CIL_A
dependencyFlags	VK_DEPENDENCY_BY_REGION_BIT
[3]	VkSubpassDependency()
srcSubpass	2
dstSubpass	3
srcStageMask	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_COLOR_ATTACHMENT
dstStageMask	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT | VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT | VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_ ELINE
srcAccessMask	VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT | VK_ACCESS_DEPTH_S
dstAccessMask	VK_ACCESS_INPUT_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_A
dependencyFlags	VK_DEPENDENCY_BY_REGION_BIT
pAllocator	NULL
RenderPass	**Render Pass 1363**

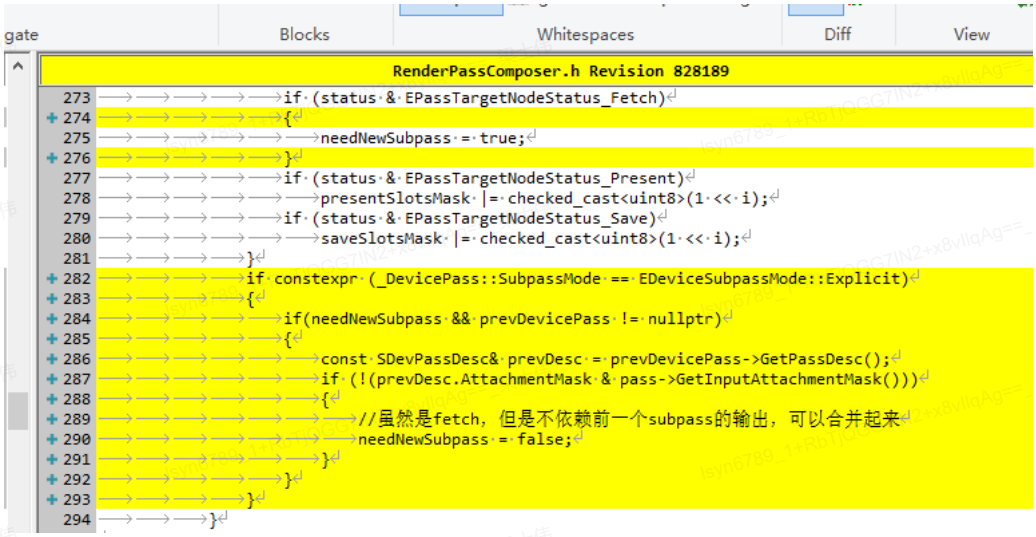
虽然这样渲染结果是正确的，但是可能会影响Subpass之间的并行度：两个完全没有依赖的Subpass，本来可以完全并行，但是现在要等的某一个Stage。
但是令人失望的是，这个还不是导致Vulkan消耗高的原因？
把Subpass之前的依赖关系整理好之后，再看ArmStreamLine的数据，还是没有明显的提升。

那系性看看有什么办法可以降低Subpass的数量：

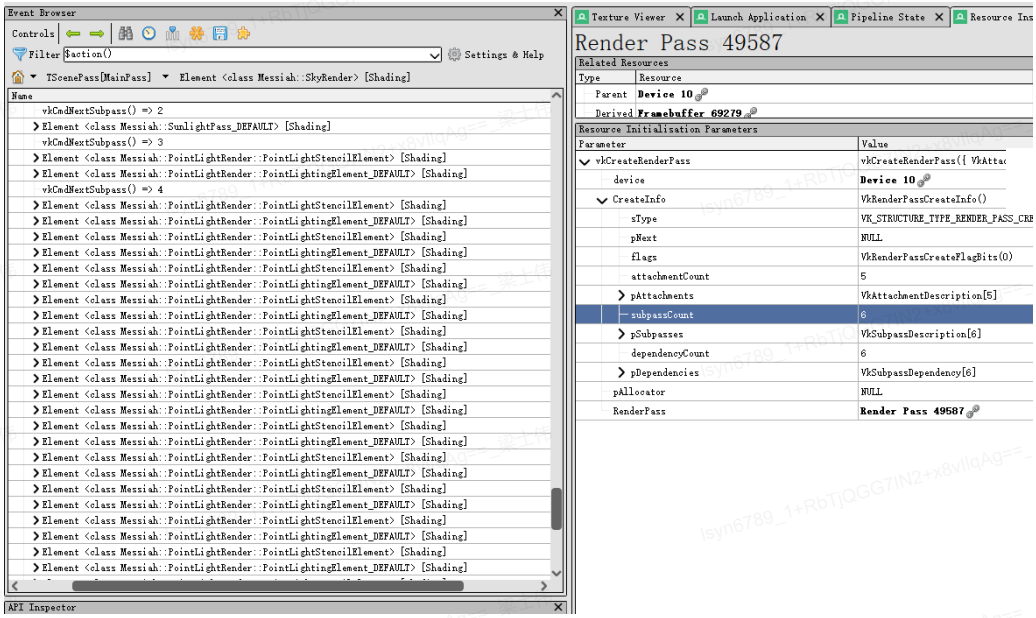
其一是发现了由于某些SceneOnly的点光错入的引入了用不到的ShadowMask，导致它和普通点光的InputAttachments不同，从而被分
Subpass。

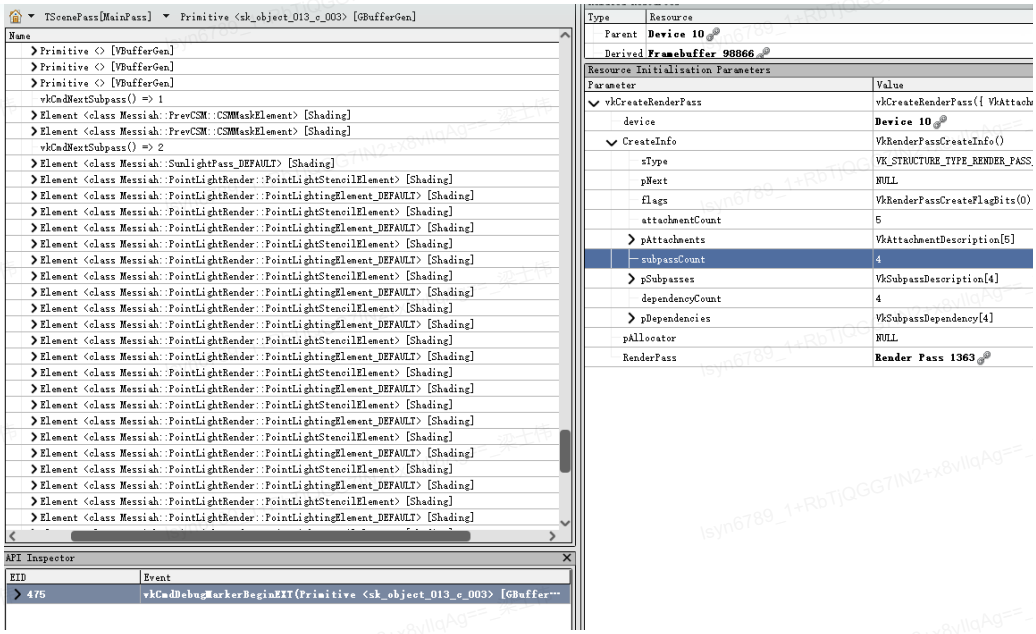
其二是两个相邻的Subpass，它们的InputAttachments完全相同，差别是是否带DepthStencil Attachment，也被分成两个Subpass。
的在没开Depth Test和Stencil Test时，绑定上DepthStencil Attachment也不会有问题，所以他们可以合并成一个Subpas。

其三同样是相邻的两个Subpass，如果上一个的Output和下一个Input没有交集，说明它们没有依赖，也是可以合并的。



在墓园出生点，合并前有6个Subpass，并且随着SceneOnly的点光增多而增多

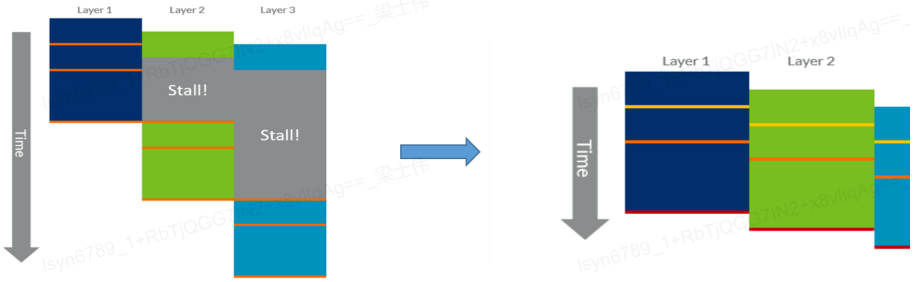




合并Subpass后GPU Active有所下降，虽然差GLES3挺远，不过也有大概5%的收益。

- Vulkan原先：10.02 mega-cycles
- Vulkan合并后：9.56 mege-cycles
- GLES：7.77 mega-cycles

后来跟MTK的工程师在交流时提到这个情况，他们认为可能是驱动导致的，并且在**Mali**最新的GPU上有优化，让我们测试一下天机900
Mali-G710 developer overview - Graphics, Gaming, and VR blog - Arm Community blogs - Arm Community
原先**Mali** GPU的Fragment Shader在并行执行中遇到需要访问Tile数据时，需要等到上一个Layer完全执行完才开始，所以会出现比较



而在最新的**Mali** G710中对此进行了优化，缩小了Tile访问的依赖范围的判定，进而减少了等待，提高了并行度。

通过测试发现，Vulkan消耗比GLES3大的状况，在天机9000上确实不存在，反而Vulkan的更好：
尽管Vulkan的fragment warps还是增加了11%，但是它的fragment active cycles减少了13%，最终的GPU Active Cycles减少了3.7%
cycles下降到23.3 mega-cycles。（以上为3帧的数据，截取于泰摩出生点）

可见在G710上应该能够放心的使用Vulkan的。而对于G77和G78等芯片，MTK的工程师建议我们可以尝试开启，虽然GPU Active Cycl
带宽确实下降不少，对减少发热降频应该有不少的贡献，所以说不准哪个更好。所以在修复了各种兼容性问题之后，让MTL针对不同芯
试。

5 MTL性能测试

整体而言，Vulkan在这些机器上的表现都是由于GLES3的，特别是在海思Kirin处理器上。

	A	B	C	D	E	F	G	H	I	J	K	L
1	机型		帧率		墓园_开	墓园_不开	黑森林_开	黑森林_不开	扎瓦因山地_开	扎瓦因山地_不开	冰雪苔原_开	冰雪苔原_不开
2		CPU型号		测试画质/分辨率	平均	平均	平均	平均	平均	平均	平均	
3	OPPO Reno 3（PDCM00）	MTK 天玑1000L	60	中	59.19712172	59.49756373	59.44735025	59.77647991	59.46873338	59.5883039	59.19261156	3
4	realme Q3 Pro（RMX2205）	天玑1100	60	高	55.2929605	57.50520695	59.28015168	57.94310332	55.08629351	55.92785451	57.23244371	5
5	红米 K40 游戏增强版	天玑1200	60	高/超清	57.34601698	56.80871349	57.01046829	56.54512132	55.69446541	55.1077088	56.16912739	55.1377395
6	realme GT Neo3（RMX3562）	MTK 天玑8100	60	极高	58.06025375	58.11489819	58.76997427	57.7890167	58.93186347	56.06019875	58.85608197	2
7	红米 K50	MTK 天玑8100	60	极高	59.02766099	58.01056646	58.03153732	58.87237251	57.77607618	58.97466519	58.68075249	3
8	红米 K50 Pro（22011211C）	MTK 天玑9000	60	/	59.10547205	58.95692138	59.23685302	58.60335906	58.90791774	58.4951476	58.9808916	1
9	华为 Mate 40（鸿蒙）	海思 Kirin 9000E	60	/	57.7385402	54.26544145	53.15186789	52.71233555	57.35585429	49.8723616	53.72596429	
10	华为 Mate 40 Pro	海思 Kirin 9000	60	/	55.11290953	45.86200888	51.23817747	50.1377017	49.72399852	47.12398671	52.04288412	1
11	华为 nova 7	海思 Kirin 985	60	/	59.10532669	56.11477491	59.00643119	58.72933141	59.0043804	56.8434797	59.16624681	57.95683535
12	三星 Galaxy S21 Ultra 5G 国际	三星 Exynos 2100	60	高/超清	47.97924917	50.28332469	47.61619188	50.70105438	44.21607539	45.20132593	46.19346842	41.39198891
13	三星 Galaxy S20 5G 英国	三星 Exynos 990	60	/	50.78716451	44.25047302	45.73762993	43.21587109			48.76276203	43.45043841
14												
15												
16												
17												
18	开代表Vulkan											
19	不开代表GLES3											
20	有底色的是Vulkan比GLES3差的											
21												

MTK工程师还建议我们接入这个拓展：OpenGL ES SDK for Android: ASTC low precision (arm-software.github.io)

ASTC在默认的情况下会先解压成16位的float，然而对于LDR的ASTC贴图这是完全没有必要的，通过ARM的这个拓展，可以指定Mali G
贴图时的精度。

How to use the extension

Decode mode is set using the a texture parameter set using the glTexParameter* functions.

```
glBindTexture(GL_TEXTURE_2D, your_astc_texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_ASTC_DECODE_PRECISION_EXT, GL_RGBA8);
```

Supported decode modes:

- GL_RGBA16F (default decode mode)
- GL_RGBA8
- GL_RGB5_E9 (requires the GL_EXT_texture_compression_astc_decode_mode_rgb9e5 extension).

可见接入非常的简单，并且收获也是非常大的，通过ArmStreamLine对比发现：
接入后提升了4x bilinear和2x trilinear filtering的利用率，降低了20%的texture active cycles，最后降低了10%的GPU Active Cycles
据，从31.7 mega-cycles下降到28.6 mega-cycles）。

7 参考

- Vulkan Multipass at GDC 2017 - Graphics, Gaming, and VR blog - Arm Community blogs - Arm Community
- Vulkan Best Practice for Mobile Developers - YouTube
- 一张图形象理解Vulkan Sub Pass - 知乎 (zhihu.com)